How browser render the page

Berat Emre Nebioğlu

supervised by
Kağan Küçük

May 11, 2020

Abstract

This article will focus on how a browser renders a page, providing insight into concepts such a reflow, repaint and a few experimental browser features useful to any frontend developer. This is an expansive topic, but here's a brief rundown.

It is suggested to check suggested reading for more information. The thing we need to take into consideration is, this article wont give you whole details just like a detour as we can presume. As It turns out, those subjects are like bottomless pit.

Contents

1 How browser understand the type of page? What do we need here?

1.1 Mime types - IANA media types, charset, viewport

1.1.1 Mime types - IANA media types

Roughly, mimetype is the key for browser to understand how to evaluate the page. If the value is given wrongly, page will be miss-evaluated. There is some mimetype we can state in the document. These are as follow;

Text/html	It is telling stream will be decipher as HTML by the browser.
Text/plain	It is telling stream will be interpreted as plain text by browser
Text/css	It is for mentioning there is CSS file.
Text/javascript	It is for mentioning there is Javascript file.
Application/octet-stream	It is for downloading dialog box.
Application/x-java-applet	It is for java-applet.
Application/pdf	It is for pdf document

[1] [2]

1.1.2 Charset

You are telling browser with that language page is evaluated. If you omit charset, browser will presume charset is ISO-8859-1.

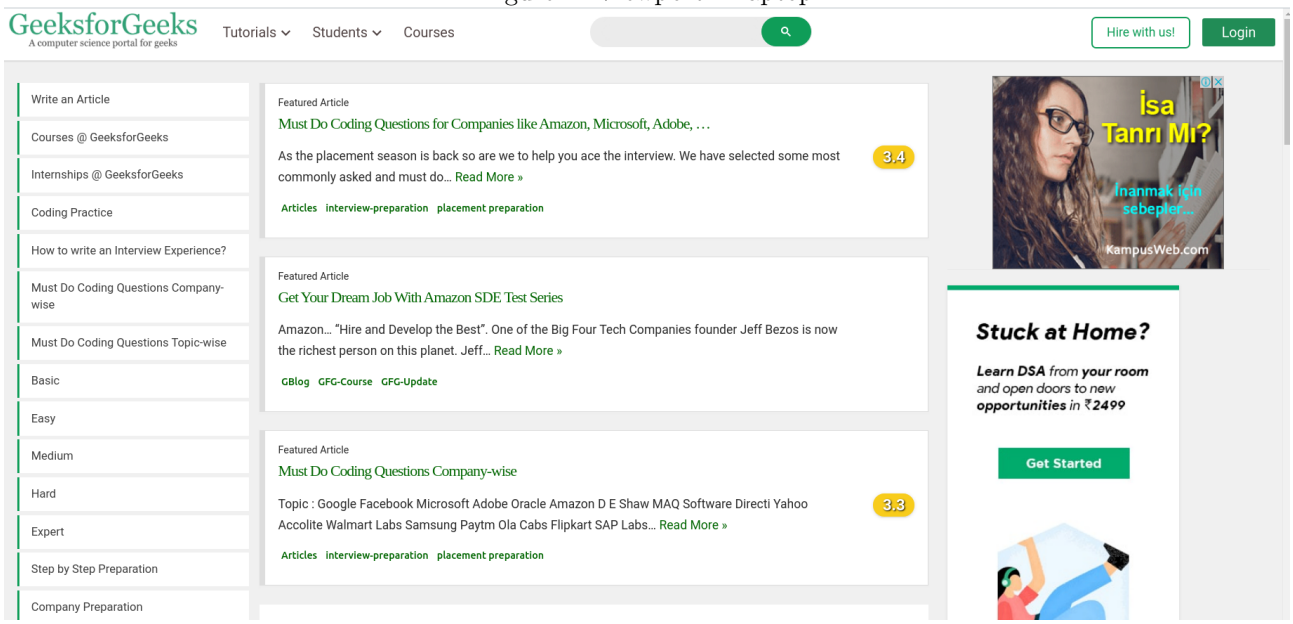
What do web browsers do if they don't find any Content-Type, either in the http headers or the meta tag? Internet Explorer actually does something quite interesting: it tries to guess, based on the frequency in which various bytes appear in typical text in typical encodings of various languages, what language and encoding was used.Internet Explorer's behaivour.

1.1.3 Viewport

- What is viewport?

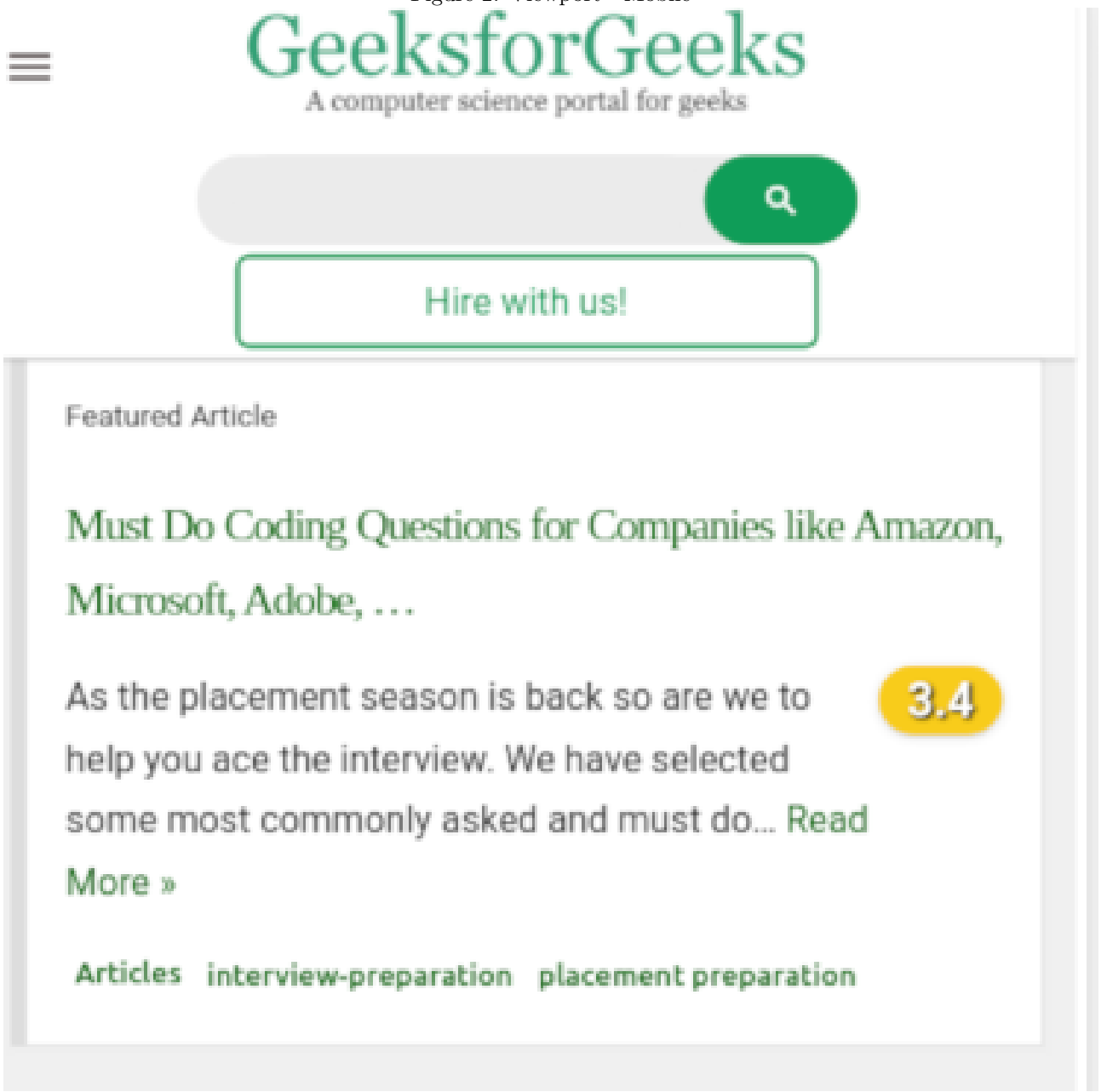
Viewport is the area that web page is visible. It can vary according to screensize. Let's take a look at how the page looks on laptop and tablet.

Figure 1: Viewport - Laptop



[1] MIME types. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
[2] Mime Type's standard can be accessed here. https://tools.ietf.org/html/rfc6838

Figure 2: Viewport - Mobile



In order to make the site responsive, you need meta viewport tag <meta name=”viewport” content= ”width=device-width, initial-scale=1.0”>

[3]

To tell browser what version of markup language will be used. Then browser will decide how page will be parsed. That also triggers what validator will validate your document.

It is the list of attribute and their corresponding explanation.

- width: Width of the virtual viewport of the device.
- height: Height of the virtual viewport of the device.
- initial-scale: Zoom level when the page is first visited.
- minimum-scale: Minimum zoom level to which a user can zoom the page.
- maximum-scale: Maximum zoom level to which a user can zoom the page.

[3] **If page is not decorated for responsive usage. You will get broken appearence.**

- user-scalable: Flag which allows the device to zoom in or out.(value= yes/no).

Figure 3: **Code Example** **Code example with viewport**

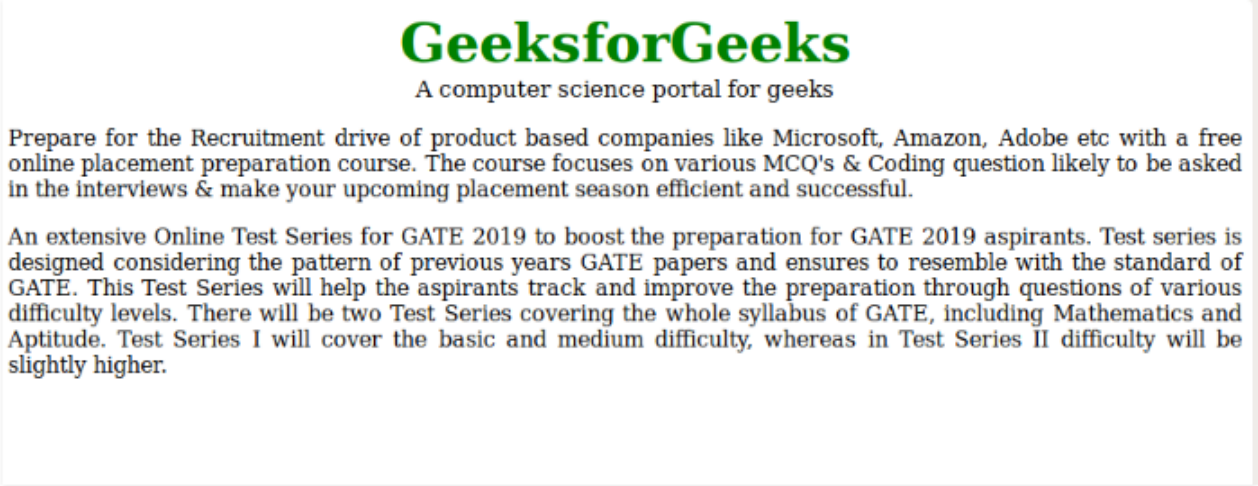
```
<!DOCTYPE html>
<html>
    <head>
        <title>GeeksforGeeks</title>
        <meta charset="utf-8" name="viewport"
        content= "width=device-width, initial-scale=1.0">
        <style>
            .gfg {
                font-size:40px;
                font-weight:bold;
                color:green;
                text-align:center;
            }
            .geeks {
                font-size:17px;
                text-align:center;
            }
            p {
                text-align:justify;
            }
        </style>
    </head>
    <body>
        <div class = "gfg">GeeksforGeeks</div>
        <div class = "geeks">A computer science portal for geeks</div>


        <p>Prepare for the Recruitment drive of product based companies like
        Microsoft, Amazon, Adobe etc with a free online placement preparation
        course. The course focuses on various MCQ's & Coding question likely
        to be asked in the interviews & make your upcoming placement season
        efficient and successful. </p>
        <p>An extensive Online Test Series for GATE 2019 to boost the
        preparation for GATE 2019 aspirants. Test series is designed
        considering the pattern of previous years GATE papers and ensures
        to resemble with the standard of GATE. This Test Series will help
        the aspirants track and improve the preparation through questions
        of various difficulty levels. There will be two Test Series
        covering the whole syllabus of GATE, including Mathematics and
        Aptitude. Test Series I will cover the basic and medium difficulty,
        whereas in Test Series II difficulty will be slightly higher. </p>
    </body>
</html>
```

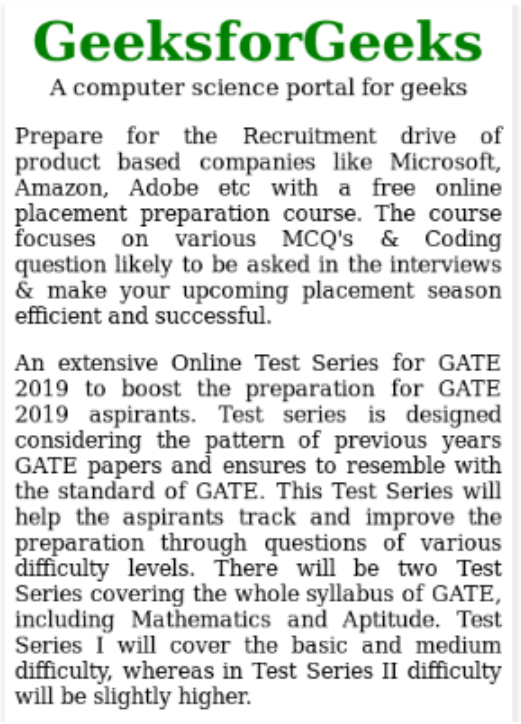
4

[4]This is inline styling it is for demonstrating purpose. Best practice is to have seperate file for style and markup.

Figure 4: **Code Example** Viewport for laptop and mobile view



Narrow Display output:



[5]

[5]You may click f12 and check dev inspector. You will see mobile icon at very very bottom and right side of pane. You can narrow and expand the page to see how element in the page is located according to current viewport.

2 DOCTYPE & MODES

[6]

2.1 DOCTYPES & Markup Styles

Browser can render the page in two mode. First one is normal mode. One way is explicitely state version of DOCTYPE. Second is let browser decide what doctpye it it.

Standard Mode

The browser render the page accoding to CSS specs.

[7]

Quirks Mode

If the browser decide, page is not structured in web standard that are published by https://www.w3.org/, browser will work on smoothing it out. Consquently you will get some unexpected result. It is always better to state explicitely the DOCTYPE at the top of document.

2.1.1 Quirks Mode and Standards Mode

In the old days of the web, pages were typically written in two versions: One for Netscape Navigator, and one for Microsoft Internet Explorer. When the web standards were made at W3C, browsers could not just start using them, as doing so would break most existing sites on the web. Browsers therefore introduced two modes to treat new standards compliant sites differently from old legacy sites.

There are now three modes used by the layout engines in web browsers: quirks mode, almost standards mode, and full standards mode. In quirks mode, layout emulates nonstandard behavior in Navigator 4 and Internet Explorer 5. This is essential in order to support websites that were built before the widespread adoption of web standards. In full standards mode, the behavior is (hopefully) the behavior described by the HTML and CSS specifications. In almost standards mode, there are only a very **small number of quirks implemented.**Quirks Mode and Standards Mode

- **How do browsers determine which mode to use?**

Figure 5: **Current DOCTYPE Standard Mode**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset=UTF-8>
    <title>Hello World!</title>
  </head>
  <body>
  </body>
</html>
```

[6] HTML Validator - https://validator.w3.org/
[7] Css specs: https://www.w3.org/Style/CSS/specs.en.html

The DOCTYPE shown in the example, <!DOCTYPE html>, is the simplest possible, and the one recommended HTML5. Earlier versions of the HTML standard recommended other variants, but all existing browsers today will use full standards mode for this DOCTYPE, even the dated Internet Explorer 6. There are no valid reasons to use a more complicated DOCTYPE. If you do use another DOCTYPE, you may risk choosing one which triggers almost standards mode or quirks mode.

Make sure you put the DOCTYPE right at the beginning of your HTML document. Anything before the DOCTYPE, like a comment or an XML declaration will trigger quirks mode in Internet Explorer 9 and older.

In HTML5, the only purpose of the DOCTYPE is to activate full standards mode. Older versions of the HTML standard gave additional meaning to the DOCTYPE, but no browser has ever used the DOCTYPE for anything other than switching between quirks mode and standards mode.

[8] [9] [10] [11] [12]

2.2 HTML - Living Standard

Here is the specs. https://html.spec.whatwg.org/multipage/semantics.html#semantics

2.3 HTML5

HTML5 is the latest evolution of the standard that defines HTML. The term represents two different concepts. It is a new version of the language HTML, with new elements, attributes, and behaviors, and a larger set of technologies that allows the building of more diverse and powerful Web sites and applications. This set is sometimes called HTML5 & friends and often shortened to just HTML5.HTML5

2.3.1 Semantics

- Sections and outlines in HTML5.
  - These are new tags -><section>, <article>, <nav>, <header>, <footer>and <aside>.
- New semantic elements.
  - These are new tags -><mark>, <figure>, <figcaption>, <data>, <time>, <output>, <progress>, or <meter>and <main>.

2.3.2 Styling

- New background styling feature.
  - Box-shadow, filters
- Animating your styles
  - Css transition ->https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions
- Typography improvement.
  - Text-overflow, @font-face
- New presential layout.
  - Using multi column layout ->https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Columns/Using_multi-column_layouts
  - Basic concept of flexbox ->https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

[13] [14]

[8] Current doctype definition is <!DOCTYPE html>For reliable CSS and JS functionality which depends on having a well formed DOM, ensure that HTML tags are properly nested and is semantically well formed.

[9] Quirks mode: https://stackoverflow.com/questions/1695787/what-is-quirks-mode/1695790#1695790

[10] Quirks mode: https://quirksmode.org/

[11] Webkit: https://github.com/WebKit/webkit/blob/a58028bd4ef8f5e5082d18e4720d3d23f0d3c54e/Source/WebCore/html/parser/HTMLConstr - Line 329

[12] Mozilla Quirks Mode Behaivour: https://developer.mozilla.org/en-US/docs/Mozilla/Mozilla_quirks_mode_behavior

[13] Full list and its explanations also the new html elements can be accessed https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. I am just taking relevant groups for the sake of relevancy.

[14] Beer in mind, animation is css transition is better for mobile performance, whereas animating in javascript is not.

3 DTD

A DTD defines the allowable nodes and allowable properties (”node attributes”), as well as defines the set of entity replacements. We can think DTD is browser’s assembly language.

DTD is metadata describing the structure of an SGML (or XML) document.

All of this is ”model”, in the MVC structure that is the web. A DTD is the definition of the structure of the model. An HTML document is then ”an instance of that model”, like a record to a database.

Figure 6: HTML is model, CSS is view and Browser is Controller MVC Figure

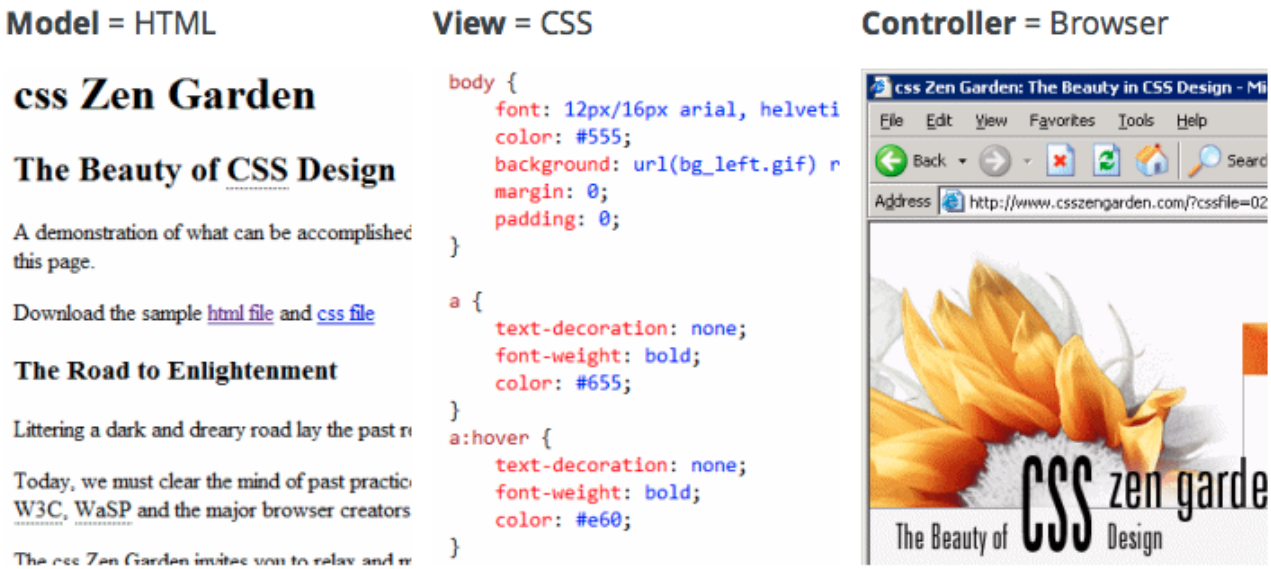


Figure 7: How dtd looks

```
<!ATTLIST TABLE \%attrs; -- \%coreattrs, \%i18n, \%events --
summary \%Text; \#IMPLIED width \%Length; \#IMPLIED border \
%Pixels;
\#IMPLIED frame \%TFrame; \#IMPLIED rules \%TRules; \#IMPLIED
cellspacing \%Length; \#IMPLIED cellpadding \%Length; \#IMPLIED
\%reserved; datapagesize CDATA \#IMPLIED \>
```

15

[15] Detailed Info: https://www.w3.org/TR/xhtml1/dtds.html#h-A2

4 Rendering

Full article is accessible here. https://html.spec.whatwg.org/multipage/rendering.html#rendering

4.1 Type of Rendering

4.1.1 Static Rendering

Static rendering is a rendering technique that you create html file and serve on the disk in each request. Nothing rendered on the fly. So it is fast and reliable. The field we use static rendering is **content website**

4.1.2 SSR - Server Side Rendering

Popular examples for Server Side Rendering are https://nextjs.org/ and **https://www.gatsbyjs.org/**. Roughly, server side rendering is a method to render your web page in backend. Browser make a request to the server and the server generates ready and valid HTML. Browser post stream to the backend. Stream is converted to HTML with browser engine.

4.1.3 When to use static rendering? When to use server side rendering?

- If the response is dynamic. **(Server Side Rendering)**
- If the response user see will change according to who is viewing it. **(Server Side Rendering)**
- If the content is static and store in seperate file. **Static Rendering**

4.1.4 CSR - Client Side Rendering

With a client-side rendering solution, when the user opens your website, his browser makes a request to the server, and the user gets a response with a single HTML file without any content, loading screen, e.t.c. It's a blank page until the browser fetches all linked JavaScripts and lets the browser compile everything before rendering the content.

4.1.5 Difference between server side rendering and client side rendering

- The main difference is that for SSR your server's response to the browser is the HTML of your page that is ready to be rendered, while for CSR the browser gets a pretty empty document with links to your javascript. That means your browser will start rendering the HTML from your server without having to wait for all the JavaScript to be downloaded and executed. In both cases, React will need to be downloaded and go through the same process of building a virtual dom and attaching events to make the page interactive — but for SSR, the user can start viewing the page while all of that is happening. For the CSR world, you need to wait for all of the above to happen and then have the virtual dom moved to the browser dom for the page to be viewable.Main difference between SSR and CSR
- CSR have blank page problem in first load. SSR does not have such problem.

4.1.6 Downside and Upside of SSR and CSR

1. SSR TTFB(Time To First Byte)is slower than CSR, because your server will have to spend the time to create the HTML for your page instead of just sending out a relatively empty response. Time to First Byte
2. SSR throughput of your server is significantly less than CSR throughput. For react in particular, the throughput impact is extremely large. ReactDOMServer.renderToString is a synchronous CPU bound call, which holds the event loop, which means the server will not be able to process any other request till ReactDOMServer.renderToString completes. Let's say that it takes you 500ms to SSR your page, that means you can at most do at most 2 requests per second.SSR throughput

16 17 18 19

4.2 Partial Rendering

The evaluation of web relies on partial rendering nowadays. This is because **refreshing whole page**, **reflowing** and **repainting** the page is costy operation.
Partial rendering is rendering tecnique for rendering part of the page instead of whole page. It is the deal of router for example. When you click button that will just show content of the page withot refreshing the page.

4.2.1 Critical Rendering

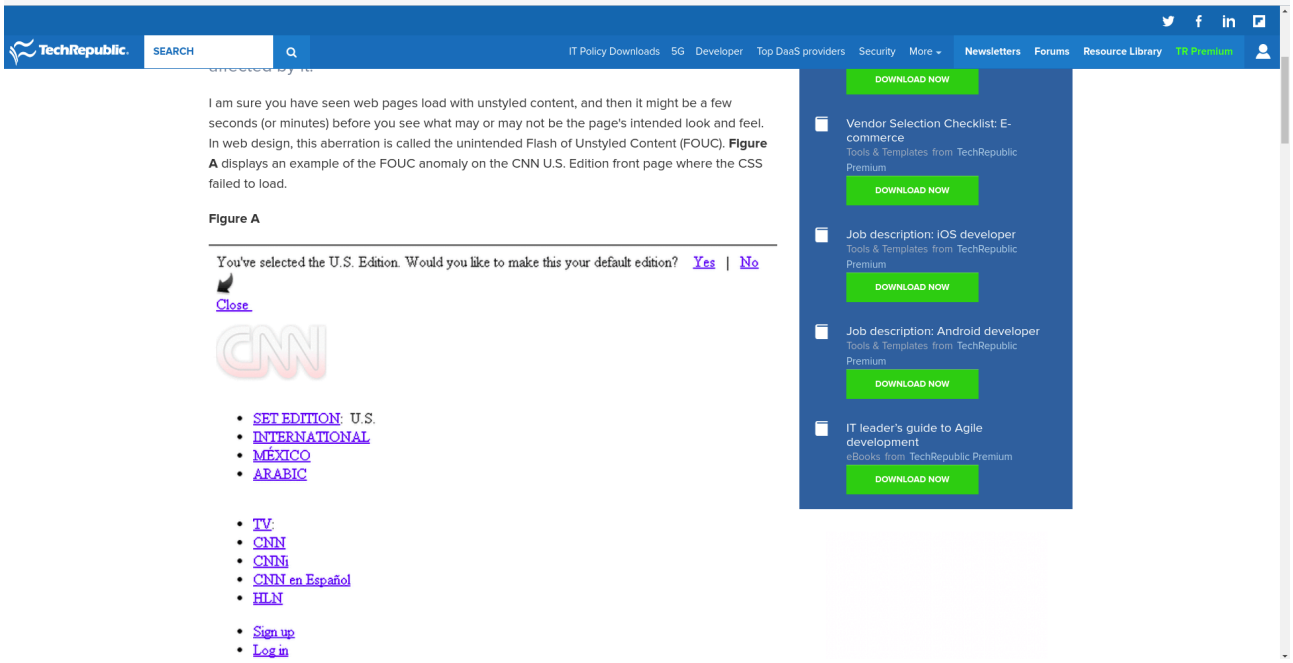
Critical rendering is about rendering viewport. If the part of the page is out of viewport it wont be rendered until you re in the coordinate of the place that is not in the viewport in the initial load.

[16] if your page relies on the JS to generate all your HTML, then client-side rendering will result in a blank page until the JS creates all the html.
[17] Time to First Byte Reading: https://en.wikipedia.org/wiki/Time_to_first_byte
[18] Throughput Explanation: Hardware strength * code efficiency + network latency = single unit throughput) * connections * requestors = total throughput
[19] Long explanation can be found here. https://gist.github.com/spinningcat/e86b14b0b3cc47c04898543bca013e64

5 FOUC - Flash of Unstyled Content

Initially browser has some value for properties. FOUC is the case when page is rendered with those initial value.

Figure 8: Flash of Unstyled Content Example - Check how menu appearence Flash Style of Content]



As you see in the image, some part of the image is without styling, more accurately part of the page is styled with predefined styles whereas other part of the page is fully customized.

5.1 Solution of FOUC

- Add some css add to head section of html. Check the figure below.

Figure 9: Piece of HTML code

```
<!DOCTYPE HTML>
<html>
    <head>
        <style
            html{
                visibility:hidden;
                opacity:0;
            }
        </style>
    </head>
    <body>
    </body>
</html>
```

20

[20]FOUC article can be found in this link. https://webkit.org/blog/66/the-fouc-problem/

5.2 Critical CSS

5.2.1 What is critical CSS?

In critical CSS you define above-the-fold content that you see in first load.

Inlining extracted styles in the <head>of the HTML document eliminates the need to make an additional request to fetch these styles. The remainder of the CSS can be loaded asynchronously.criticalcss

if you have a scrolling one page website, the "above the fold" content is the stuff at the top, not requiring any action like scroll.

Above the fold is the upper half of the front page of a newspaper or tabloid where an important news story or photograph is often located. Papers are often displayed to customers folded so that only the top half of the front page is visible. Thus, an item that is "above the fold" may be one that the editors feel will entice people to buy the paper. Alternatively, it reflects a decision, on the part of the editors, that the article is one of the day's most important. By extension, the space above the fold is also preferred by advertisers, since it is the most prominent and visible even when the newspaper is on stands.Above The Fold

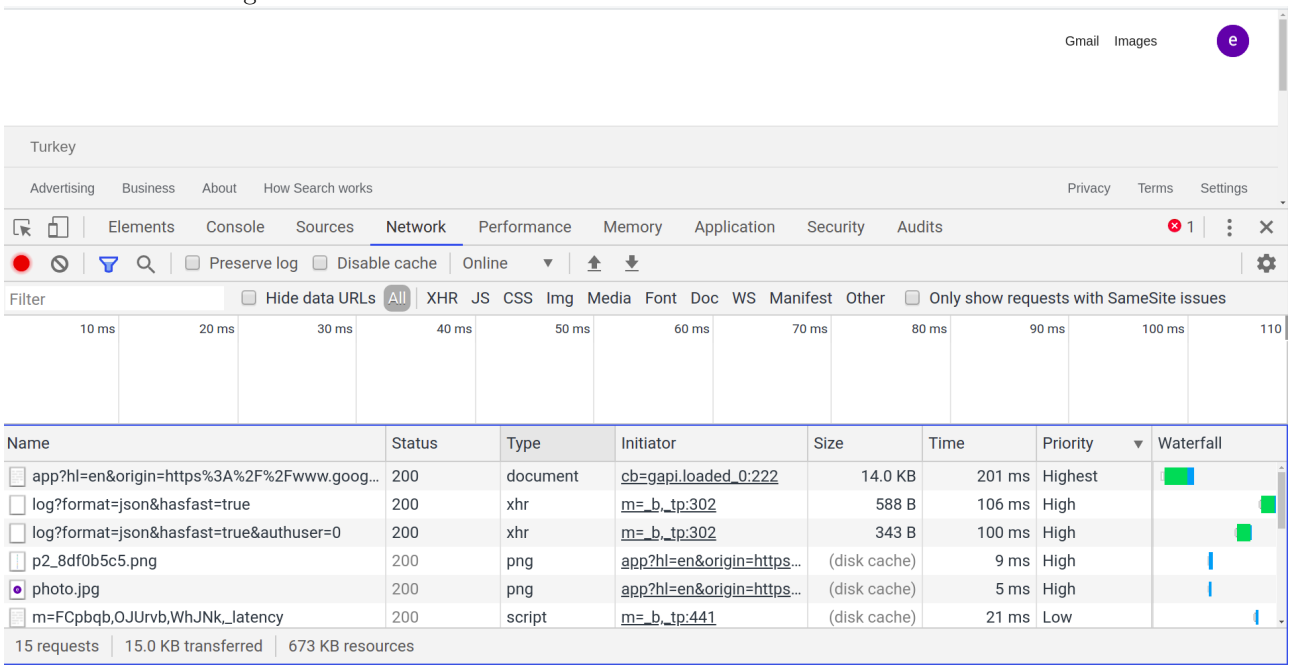
21 22

[21] the "fold" is relative — different devices will display different amounts of content on initial load
[22] Google PageSpeed Insight: https://developers.google.com/speed/pagespeed/insights/

6 Priorities of assests in the page

6.1 Chrome Dev Tools

Figure 10: It can be examined in Chrome Dev Tools. Chrome Dev Tools

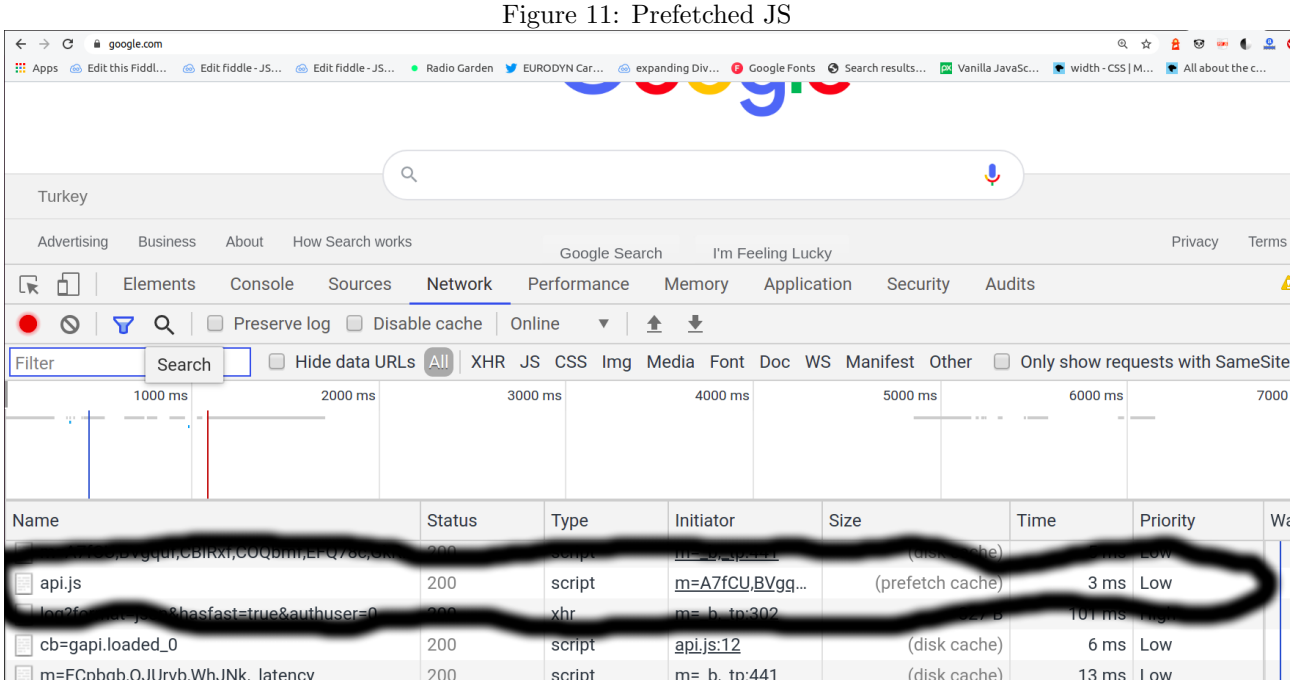


6.2 Prefetch

6.2.1 Definition

<link rel="prefetch">, browser will download resource and cache it (script, stylesheet). It is beneficial to use that feature, if you use resource again and again. Priority is low.

Figure 11: Prefetched JS



6.2.2 Syntax

<link rel="prefetch" href="/style.css" as="style" />

6.2.3 Type of Prefetch

- Link Prefetching

The prefetch link relation type is used to identify a resource that might be required by the next navigation, and that the user agent SHOULD fetch, such that the user agent can deliver a faster response once the resource is requested in the future.Link Prefetching

- DNS Prefetching

A DNS or domain name server converts IP addresses in readable website URLs such as yourwebsite.com. Whenever a user requests an asset being hosted on a particular domain they must perform a DNS lookup and find which domain name that IP address belongs to. This process takes time and the most DNS lookups that are required, the longer your visitors will be waiting for a page to load.DNS Prefetching

[23] [24]

6.3 Prerender

Prerender is also render the page besides downloading resources that are requested.

[25]

6.4 Preload

6.4.1 Definition

<link rel="preload">

6.5 Explanation

Preload is used for loading critical resources. This is the list that can be preloaded.

1. Scripts
2. External CSS
3. Images from ¡img>tags

6.6 Advantages of Preload (Check Link #9 in the inspired source section.)

1. Preload does not block **onload** event.
2. Early loading of fonts.
3. Dynamic loading without execution.
4. Markup based async loader.
5. Responsive loading.

[26] [27] [28] [29]

[23] You should note that only cacable source can be prefetched.
[24] Browser Support: https://caniuse.com/#search=prefetch
[25] Prerender Browser Support: https://caniuse.com/#search=prerender
[26] There are limits to how many files a browser can download in parallel. The limits vary between browsers and depend on many factors, like whether you're downloading all files from one or from several different servers and whether you are using HTTP/1.1 or HTTP/2 protocol. To render the page as quickly as possible, browsers optimize downloads by assigning priority to each file. To figure out these priorities, they follow complex schemes based on resource type, position in the markup, and progress of the page rendering.Factors in Limitation
[27] Speculative Parsing: https://developer.mozilla.org/en-US/docs/Glossary/speculative_parsing
[28] Advantage of preload is to give a way to user to define loading mechanism. Performance Mechanism
[29] Preload Browser Support: https://caniuse.com/#search=preload

6.6.1 Syntax

<link rel=”preload” href=”/style.css” as=”style” />

6.6.2 What ”as” attribute include.

- Script
- Style
- Image
- Media
- Document

6.7 Lazy Loading

Lazy-loading is a performance optimisation technique that makes it possible to load only the required sections of your web/mobile pages on demand instead of in bulk.Definition of Lazy Loading

6.8 Intersection Observer API

In old times, you are triggering lazy-load with some events such as **scroll**, **resize**, **orientationchange** These were not performant and **intersection oberserver api** comes in play.

It provided us a way to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document’s viewport. More importantly, it provides us a callback function that will fire when the element we are observing gets into the viewport.Intersection Observer API

Figure 12: Intersection Observer API Example

```
<div>
Load count: <span>0</span>/4
</div>
<img class="lazy" data-src="https://placekitten.com/100/100">
<img class="lazy" data-src="https://placekitten.com/105/105">
<img class="lazy" data-src="https://placekitten.com/110/110">
<img class="lazy" data-src="https://placekitten.com/115/115">

img {
  display: block;
  width: 300px;
  height: 300px;
}
div {
  position: fixed;
  right: 4px;
  top: 4px;
}

const imgs = [...document.querySelectorAll("img.lazy")];
let count = 0;

const updateCount = count => {
    document.querySelector("span").textContent = count;
}

const lazyImageObserver = new IntersectionObserver(function(entries, observer) {
  entries.forEach(function(entry) {
    if (entry.isIntersecting) {
        setTimeout(() => {
        const lazyImage = entry.target;
        const bound = lazyImage.getBoundingClientRect();
        if (bound.top <= window.innerHeight && bound.bottom >= 0) {
          lazyImage.src = lazyImage.dataset.src;
          lazyImage.classList.remove("lazy");
          lazyImageObserver.unobserve(lazyImage);
          updateCount(++count);
        }
      }, 300);

    }
  });
});

imgs.forEach(function(lazyImage) {
  lazyImageObserver.observe(lazyImage);
});
```

30

[30] Original Code: https://jsfiddle.net/hellyeah/cphfq79y/

6.9 Native Lazy Loading

Chrome loads the images with different properties by respecting wherebout it is located. In chrome 76, it is possible to use **loading** to delay loading of image or iframe.

Figure 13: Laziliy Loaded

```
<img src="image.png" loading="lazy" alt="..." width="200" height="200">
<iframe src="https://example.com" loading="lazy"></iframe>
```

6.9.1 Loading Attribute Options

1. **Lazy:** Lazily load image when it is in viewport.
2. **Eager:** Load right away.
3. **Auto:** Browser will decide.

31 32 33 34
35

[31] Original Code: https://jsfiddle.net/hellyeah/kxc7dyor/1/
[32] Adding width and height with inline style will prevent reflow.
[33] For refreshers, lazy-loading is a performance optimisation technique that makes it possible to load only the required sections of your web/mobile pages on demand instead of in bulk. What this means is that for each time users scroll through your application, contents are served on demand (until they get into the viewport.Definition of Lazy Load
[34] Intersection Observer Vue Directive: https://gist.github.com/kaankucukx/d0ac401e162f911e3b88f159054e7fb3
[35] Chrome 76 has loadimg attribıte to lazy-load resources. Lazy Load Feature on chrome

6.10 Old style lazy loading

Figure 14: Code Example

```
<script>
    window.lazySizesConfig = window.lazySizesConfig || {};
  window.lazySizesConfig.rias = window.lazySizesConfig.rias || {
    prefix:  'https://res.cloudinary.com/eitanpeer/image/fetch/q_auto,f_auto,w_{width}/',
    absUrl: true
  };

    //only for demo in production I would use normal expand option
    window.lazySizesConfig.expand = 9;
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/lazysizes/3.0.0/plugins/rias/ls.rias.min.js" async></script>
<script src="//cdn.jsdelivr.net/g/lazysizes(lazysizes.min.js)" async=""></script>

<div id="container" class="container">
</div>

*, :before, :after {
  box-sizing: border-box;
}

.container {
  img {
    width: 50%;
  }
}

.lazyloading {
  background: #f7f7f7 url(//afarkas.github.io/lazysizes/assets/imgs/loader.gif) no-repeat center;
  min-height: 60px;
}

const imageSources = [
  'https://images.unsplash.com/photo-1493020258366-be3ead1b3027',
  'https://images.unsplash.com/photo-1492974886515-a4036e79e639',
  'https://images.unsplash.com/photo-1476901192299-9a86d0454988',
  'https://images.unsplash.com/photo-1493018245248-737b8728173f'
]

imageSources.forEach(src => {
  const imgHtml = `
  <div class="img-wrapper">
    <img src="" data-src="${src}" data-sizes="auto" class="lazyload">
  </div>`;
  document.getElementById("container").innerHTML += imgHtml;
});
```

36

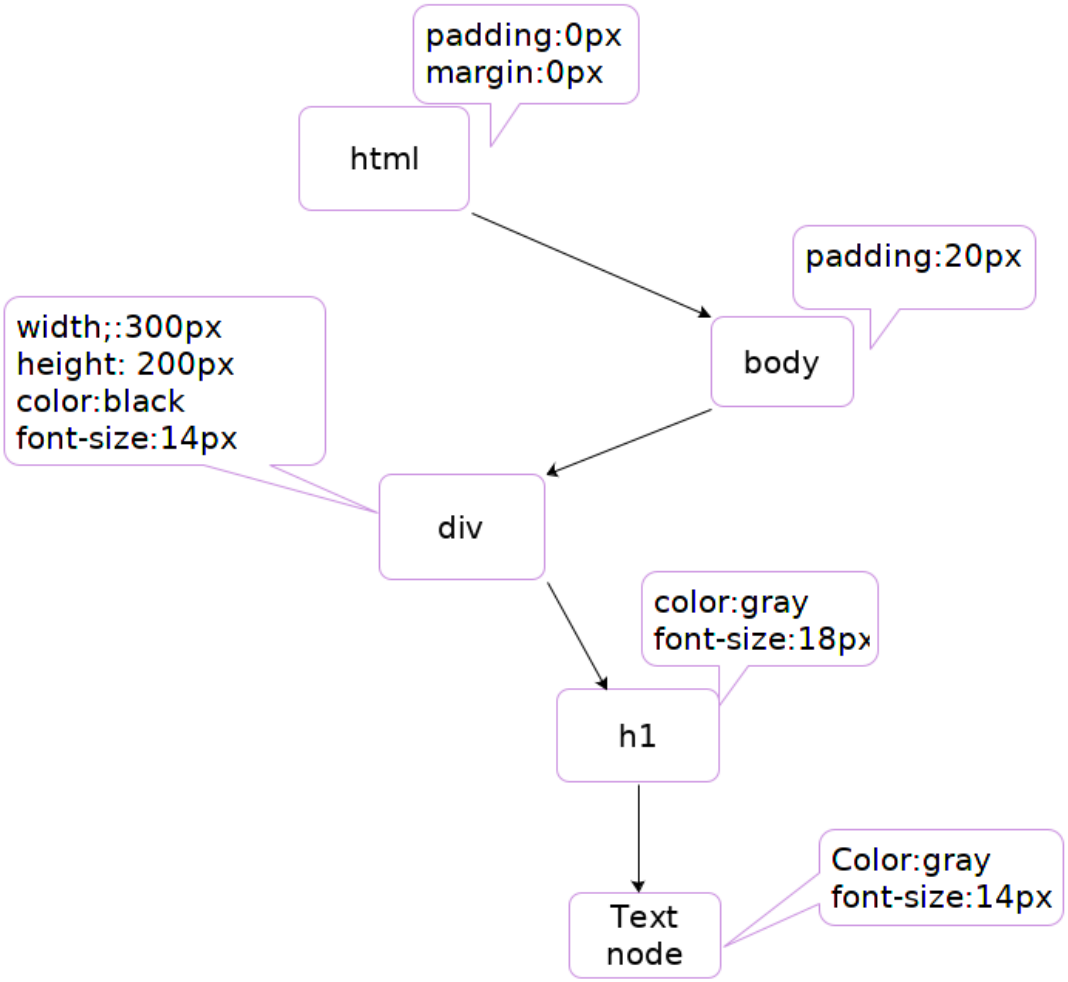
[36] Original Code: https://jsfiddle.net/hellyeah/kxc7dyor/1/

7 Rendering Content

7.1 What is render tree?

Render tree is tree structure that is constructed with the combination of DOM and CSSOM. Browser will calculate the layout of each visible element and paint them.

Figure 15: Render Tree Visual



The steps of rendering process is

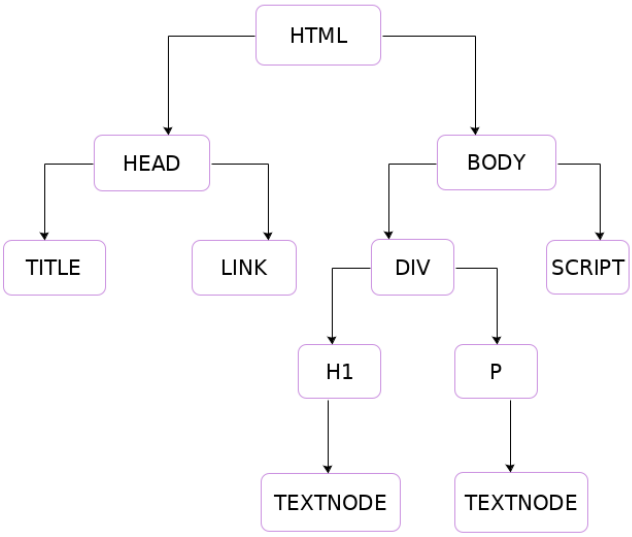
- Read text html and construct a dom
- Process the css
- Construct CSSOM
- Construct render tree (combining DOM and CSSOM)
- Then print it to browser
- Layout operation
- Paint operation
- Compositting operation

7.2 What is DOM (Document Object Model)

When the browser reads html code, a node is created. A node is javasript object that has some property. We can think dom as nary tree. One is root the parent at the top and others are child, siblings etc.
Javasript does not understand DOM.

Figure 16: DOM Visual

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Title</title>
  <link rel="stylesheet" ref="style.css">
</head>
<body>
  <div>
      <h1>h1 text</h1>
      <p>p text</p>
  </div>
  <script src="script.js"><script>
</body>
</html>
```



7.3 What is VDOM?

VDOM is a layer that some frameworks like angular,react, vue etc. do some operation like node insertion, or update component etc. Advantage of working VDOM compared to DOM, in VDOM you do your work and apply VDOM to DOM just once. It has some performance gains. Whereas in DOM, when you do insertion, browser will reflow the dom from zero. Please check below;

1. The Virtual DOM is an abstraction of the HTML DOM. It is lightweight and detached from the browser-specific implementation details.
2. With VDOM, you only manipulate DOM once. It is performance-efficient.

Figure 17: React functional component and Simple Redux

```
import React from "react";

type props = {
    smth: String;
    smAction: () => void;

}
export const DummtFCComponent: FC<Props> = props => <div></div>
```

7.4 Why DOM?, Why VDOM? Prons and cons

7.4.1 DOM

- It is tree structure eay to traverse. It is often time a massive structure.

- We need to find node to do some changes value or binding event. We need to go through from up to down.
- Maniplating DOM in every single moment is so costy. That means **reflow** and **repaint**

7.5 Language & Framework use DOM or VDOM

- Vanilla Javascript (DOM) ->https://eloquentjavascript.net/
- Jquery (DOM) ->https://jquery.com/
- Svelte (DOM) ->https://svelte.dev/
- Angular (VOOM) ->https://angular.io/
- ReactJS (VDOM) ->https://reactjs.org/
- VueJS (VDOM) ->https://vuejs.org/

[37]

7.6 What is CSSOM

The interaction between HTML and CSS can be maintained with;

1. Id selector
2. Class selector
3. Tag Selector

After constructing the DOM, the browser reads CSS from sources and construct CSSOM. It is like DOM. Element in CSSOM has CSS rules.
Each browser comes with set of CSS rules. The external css rules are overridden and applied. While css rules are applied **the specificity** rules matters that are illustrated below.

- Class has priority then id.

7.7 What scss, sass, less brings extra to web development world?

- Variable
- For loop
- Mixins
- Pre-defined methods
- Import
- Ability to write methods

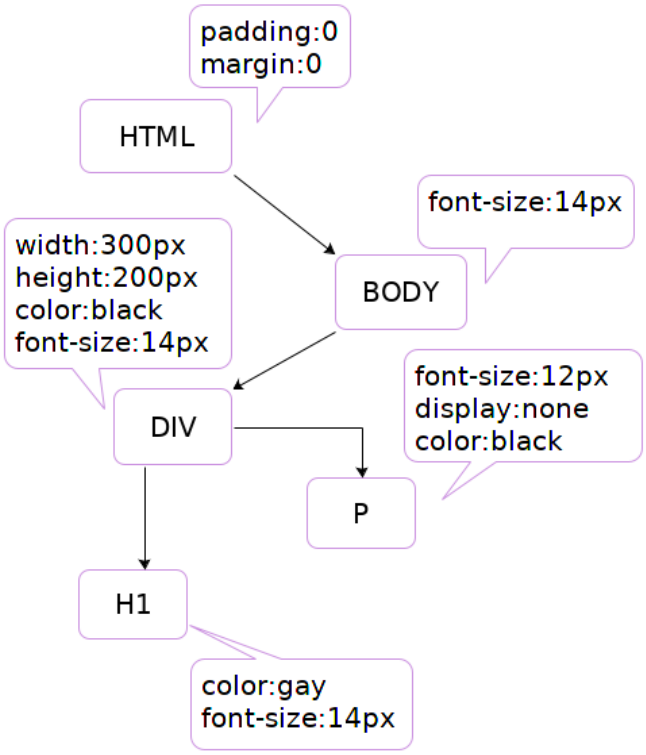
7.8 Advantages of CSS preprecessor.

- Nested syntax.
- Ability to define variables.
- Ability to define mixins.
- Mathematical functions.
- Operational functions (such as “lighten” and “darken”)
- Joining of multiple files.

[37] ReactJS vs Svelte: https://medium.com/javascript-in-plain-english/svelte-vs-react-first-impression-1ce5d3ee6889

Figure 18: CSSOM Visual

```
html {
    padding: 0;
    margin: 0;
}
body {
   font-size: 14px;
     }
.container {
   width: 300px;
   height: 200px;
   color: black;
}
. container > h1 {
   color: gray;
}
.container > p {
   font-size: 12px;
   display: none;
}
```



7.9 Disadvantages of preprecessor.

- Preprocessor needs to be compiled so it slows down development and make debugging hard.
- Big CSS file
- Using Sass may cause of losing benefits of browser's built-in element inspector.

Dezavantajla ilgili kaynak paylaşabilirsen iyi olur.

7.10 Spesificity

Specificity measure with a weight of CSS selectors. If an element will have class and id together and id and class have same set of rules in the iah yeahd will be applied. If selectors have same weight, last selector you define in CSS file will be applied.

Selector Types

- Type Selectors like h1, p or div
- Class selector .class or [class=class]
- Id selector .id or [id=id]

Figure 19: Specificity #1

HTML

```
<button id="id" class="class">
      Spesicificity
</button>
```

CSS

```
#id
{
     background-color:blue;
}
.class
{
    background-color:green;
}
```

if you have and in your CSS you have #demo { colour: red } .example { background: lime }, even though the #demo selector is stronger, that element is going to end up with the properties from both rules because both rules apply

where specificity matters is when you have #demo { colour: red } and .example { colour: lime } trying to set the same property. In this case they still also both apply, but since #demo is stronger than . example, its colour value overrides the weaker rule.

selectors don't battle each other, they just assign different 'weight' to the values you're specifying and a stronger selector can huld properties that override properties applied to the same element by a weaker selector.

Figure 20: Specificity #2

HTML

```
<button id="id" class="class">
      Spesicificity
</button>
```

```
[class=class]
{
    background-color:green;
}
[id=id]
{
   background-color:blue;
}
```

Additional note if you select element with [class=class], [id=id]. Something will happen differently. Because they are attribute selectors and they have equal weight.

- Specficity determines which css rules will be applied.
- Every selectos have its own weight.
- If two selectors have same set of rules. Stronger selector will defeat weaker selectors.
- This site can give you more information about Specificity
- If selectors are equally strong. Last one will be applied.
- When selectors have an unequal specificity value, the more specific rule is the one that counts.

8 Advanced Content

8.1 Browser Rendering Optimizations for Front-end Development

We live in an age where the importance of delivering web services at optimal speed can't be overemphasized. As the payload transmitted by web applications increase, developers must adopt best practices to ensure that data packets are delivered almost instantaneously, hence providing users with an overall exemplary experience.

Some of the widely adopted best practices in web development today are image compression, code minification, code bundling (with tools such as Webpack), and so on. These practices already have the effect of improving user satisfaction, but it is possible to achieve more when the developer understands the underlying steps that guide the rendering of web applications to the DOM. Important of Rendering Optimization

8.1.1 What does into frame

1. Browser will get stream that includes HTML and CSS as stream.
2. HTML is parse into DOM and CSS is parsed into CSSOM they combine info render tree.
3. Each node of render tree is a frame.

8.1.2 App Lifecycle

App Lifecycle

1. **Load:** Before a user can interact with a web application, it has to be loaded first. This is the first stage in the app lifecycle and it is important to aim at reducing (ideally at 1s) the load time to the smallest number possible.
2. **Idle:** After an application is loaded, it usually becomes idle; waiting on the user to interact with it. The idle block is usually around 50ms long and provides the developer with the opportunity to do the heavy lifting, such as the loading the assets (images, videos, comments section) that a user might access later.
3. **Animation:** When the user starts interacting with the application and the Idle stage is over, the application has to react properly to user interaction (and input) without any visible delay.
4. **Response**

8.1.3 Layout Creation

A major performance bottleneck is layout thrashing. This occurs when requests for geometric values interleaved with style changes are made in JavaScript and causes the browser to reflow the layout.Reflow problem

8.1.4 Painting of Screen Pixels

Painting occurs when the browser starts filling in screen pixels. This involves drawing out all visual elements on the screen. This is done on multiple surfaces, called layers. Painting

8.1.5 Layer Composition

The browser engine does some layer management by first considering the styles and elements and how they are ordered, then tries to figure out what layers are needed for the page and updates the layer tree accordingly. Layer Composition

8.1.6 Layout Trashing

Layout is what browser figures out following;

- Their size
- Location in the page

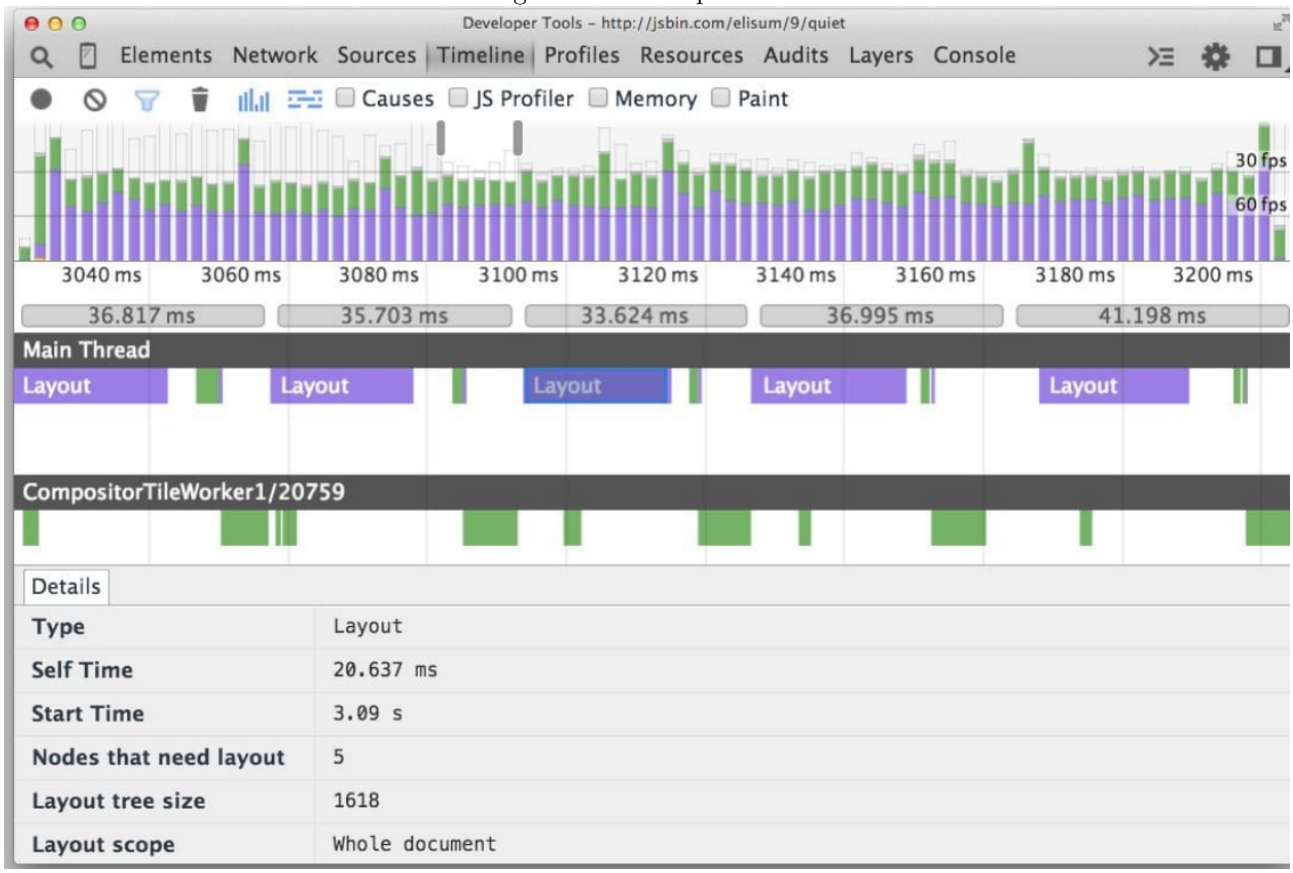
This process is called layout in chrome, opera, safari, internet explorer and reflow in firefox. The biggest concern here about composing is as follow;

- The number of element that requires layout.
- The complexity of layout.

38 39 40 41 42 43 44

If it's not possible to avoid layout then the key is to once again use Chrome DevTools to see how long it's taking, and determine if layout is the cause of a bottleneck. Firstly, open DevTools, go to the Timeline tab, hit record and interact with your site. When you stop recording you'll see a breakdown of how your site performed: LayoutTrashing

Figure 21: Developer Tools



8.2 Css triggers

That some changes can trigger more changes. Subsequent changes to the layout, that is. You typically want to minmize changes to the DOM. This is most important for complex layouts, where it can severely impact UI responsiveness.

45

[38] If you design the page that requires layout "that step include calculating geomery of the page, draw elements, replace each of them to correct place", will cost you sometime. So it is better to avoid layout (reflow)
[39] Avoid forced synchronous layouts and layout thrashing; read style values then make style changes. Layout Trashing
[40] There is a problem when elements are in same layer. To fix this problem use will-change:transform. will-change:transform
[41] Visual change that is done by Javascript ot CSS will initialize series of events such as recalculate the style.
[42] If change in the geometry, browser will create a new layout, repaints elements that are effected and re-composite them.
[43] Property like font-color, background-color only triggers re-paint.
[44] Property like position may be the reason of chage of all layout.
[45] Detaied Info: https://csstriggers.com/

8.3 Rendering sequence

8.3.1 Layout

CSSOM and DOM are combined into render tree Render tree is used for computing the layout each visible elements. Briefly, layout is about drawing each element with its geometry, and replace those to location in window.

8.3.2 Paint

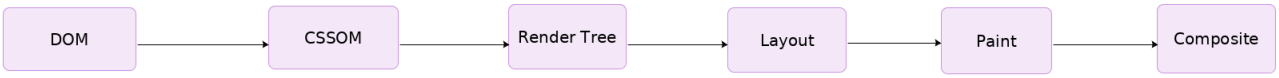
Paint each pixel in the web page.

8.3.3 Compositting

When sections of the document are drawn in different layers, overlapping each other, compositing is necessary to ensure they are drawn to the screen in the right order and the content is rendered correctly.
Compositting can be the reason of **reflow** and reflow can spark **repaint**.

8.3.4 Rendering Steps Diagram

Figure 22: Critical Rendering Path



8.4 Reflow

8.4.1 Definition

A reflow computes the layout of the page. A reflow on an element recomputes the dimensions and position of the element, and it also triggers further reflows on that element's children, ancestors and elements that appear after it in the DOM. Then it calls a final repaint. Reflowing is very expensive,

8.4.2 What triggers reflow

- Insert, remove or update an element in the DOM
- Modify content on the page, e.g. the text in an input box
- Move a DOM element
- Animate a DOM element
- Take measurements of an element such as offsetHeight or getComputedStyle
- Change a CSS style
- Change the className of an element
- Add or remove a stylesheet
- Resize the window
- Scroll

8.4.3 Css property that cause reflow

Position is the one property that triggers reflow. The alternative usage of position is **transform**. **Transform** wont trigger reflow.

8.5 Repaint

8.5.1 Definition of repaint

Repaint is colorizing each element in the page. It happen after whole page is drawn.

8.5.2 Css property that cause reflow

Background color ->Bura içeriğini nasıl daha güzel verebilirim yönlendirebilirsen sevinirim.

8.5.3 Transform versus position:absolute

The drawing order consist of The layout layer, paint layer, and compositor layer.

Changing left or margin will trigger a redraw in layout layer (which, in turn, will trigger redraws in the other two layers) for the animated element and for subsequent elements in DOM.

Changing transform will trigger a redraw in compositor layer only for the animated element (subsequent elements in DOM will not be redrawn).

Translate happens when all the layout process complete, further it even already painted, what is remaining is a matter where to put the element, so it has no interaction with the existing layout.

Figure 23: Position and Transform

```
<div class="coordinates"></div>
<div class="box move"></div>
<div class="box scale"></div>
<div class="box rotate"></div>
<div class="box skew"></div>
<div class="box move-and-rotate"></div>

.coordinates{
    position: absolute;
    left: 100px;
    top: 20px;
    border-left: 1px dotted black;
    border-top: 1px dotted black;
    height: 750px;
    width: 500px;
}

.box{
    margin: 20px 100px;
    background-color: red;
    width: 100px;
    height: 100px;
}

.move{
    transform: translate(10px, .5em);
}

.scale{
    transform: scale(2, 0.5);
}

.rotate{
    transform: rotate(30deg);
}

.skew{
   transform: skew(15deg);
}

.move-and-rotate{
    transform: translate(100px, 50px) rotate(50deg);
}
```

46 47

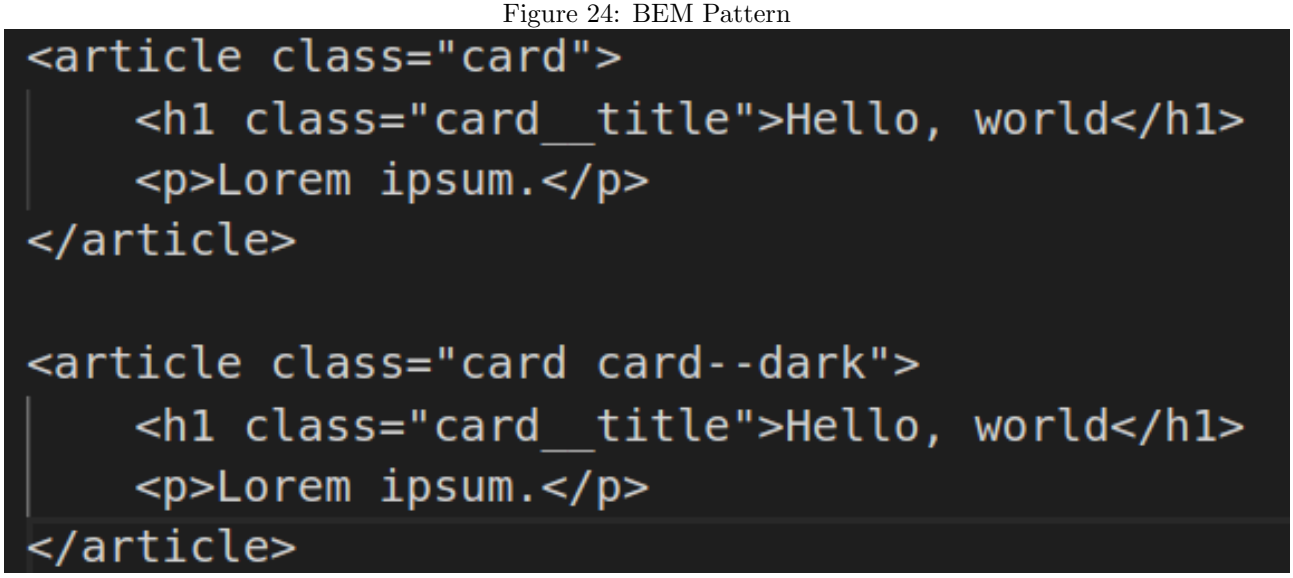
[46] Original Code: https://jsfiddle.net/hellyeah/6mu2abhs/
[47] More insights: https://developers.google.com/web/fundamentals/performance/rendering/

8.6 Organizing CSS

8.6.1 Block Element Modifier - BEM

- BEM class starts with a block, which is an object name.
- Children of that block, you add an element, separating it with two underscores.
- you can modify any class (block or element) by adding a modifier, separated with two hyphens.

Figure 24: BEM Pattern



[48]

8.6.2 Object Oriented CSS - OOCSS Organizing-css-oocss-smacss-and-bem

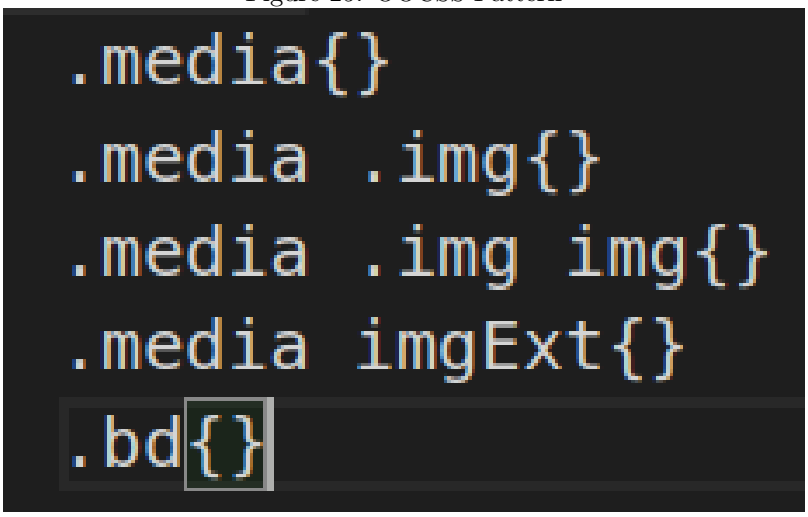
OOCSS is a programming paradigm. OOCSS stands for Object Oriented CSS, so it's best understood in the context of Object Oriented programming: classic (spaghetti) CSS vs. OOCSS is a bit like procedural (spaghetti) backend code vs. Object-Oriented backend code.

OOCSS focuses on flexible, modular, swappable components that do One Thing Well. OOCSS focuses on the single responsibility principle, separation of concerns, and much more of the foundational concepts of Object Oriented Programming.

For a great introduction to OOCSS, this post on the OOCSS Media Object (written by the/one of the people behind OOCSS) shows an example of what a CSS object looks like, and some of the benefits of using one.

[48] Detailed Info: BEM - http://getbem.com/introduction/

Figure 25: OOCSS Pattern



8.6.3 Scalable and Modular Architecture CSS - SAMACSS Organizing-css-oocss-smacss-and-bem

SMACSS stands for Scalable and Modular Architecture for CSS. It's a book and a methodology for writing CSS (created by Jonathan Snook), but its most significant and influential aspect is its organizational system, which is designed to provide a set of buckets into which CSS should be organized. To learn more, check out the SMACSS web site.

8.6.4 styling methodology for component-based UI development - SUITCSS

SUITCSS is another pattern for organizing CSS in semantic way. Let's examine the pattern with code example.

Figure 26: SUITCSS Pattern

```
<header class="Header">
    <a href="#" class="Header-link">
        <img src="..." class="Header-logo" />
    </a>
    <nav class="Nav Nav--topMenu Header-nav">
        <a href="#" class="Nav-link is-active">...</a>
        <a href="#" class="Nav-link--red">...</a>
    </nav>
</header>

Header is component. This is written with PascalCase.
Link in header-link is a descendant that is written with camelCase and -.
--red is a css modifier. It is written with --.
```

8.6.5 Normalize CSS

Normalize CSS is out there to maintain consistency amongst browsers. It supports wide range of browsers. That also normalize HTML5 elements, typography, list, embedded content, form and tables.

- Preserve useful browser defaults.
- Normalize styles for a wide range of HTML elements.

- Correct bugs and common browser inconsistencies.
- Improve usability with subtle improvements.

[49] [50] [51] [52]

8.7 Reasons not to use IDs in CSS

Here is couple of reasons to use class over id in CSS.

- Class specificity is lower than ID specificity.
- Classes can be reused.
- A consistent convention.
- An element can have several classes, but only one ID.

8.8 HTML <template>tag

with <template>you can build a web component that wont be rendered in first page load.

Figure 27: HTML template tag example

```
<template>
    <div class = "container">
        <h1> Hello World! </h1>
        <p> Container div will be ignored in the first time.
        That will be injected later with the help of javascript.
    </div>
</template>
```

8.9 Webworkers

Webworkers async, isolated.

Webworkers is the workers that runs scripts in the background. Web Workers allow you to do things like fire up long-running scripts to handle computationally intensive tasks, but without blocking the UI or other scripts to handle user interactions. Webworker is isolated thread.

The simple usage of webworkers is computationally expensive task. We can take a look at simple webworkers example in the next section.

[53] [54]

8.9.1 Webworkers real world example

- Worker() constructor create aworker and returns it.
- Onmessage event handler recieve message from handler.
- postmessage() method send the message to worker when prime number is found.

[49] Article: http://nicolasgallagher.com/about-normalize-css/
[50] Normalize css is alternative to css reset with some difference that are mentioned here.
[51] Normalize : https://cdnjs.com/libraries/normalize
[52] Maintainable CSS: https://www.integralist.co.uk/posts/bem/#4
[53] Detailed Info: Web Workers - https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API
[54] More insights can be found here. https://html.spec.whatwg.org/multipage/workers.html#workers

Figure 28: Piece of HTML code Web Workers - HTML]

```html
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>Worker example: One-core computation</title>
 </head>
 <body>
  <p>The highest prime number discovered so far is: <output id="result"></output></p>
  <script>
   var worker = new Worker('worker.js');
   worker.onmessage = function (event) {
     document.getElementById('result').textContent = event.data;
   };
  </script>
 </body>
</html>
```

Figure 29: Piece of HTML code Web Workers - Javascript]

```javascript
var n = 1;
search: while (true) {
  n += 1;
  for (var i = 2; i <= Math.sqrt(n); i += 1)
    if (n % i == 0)
      continue search;
  // found a prime!
  postMessage(n);
}
```

8.10 Webpack

Webpack is a module bundler. Webpack can take care of bundling alongside a separate task runner. However, the line between bundler and task runner has become blurred thanks to community developed webpack plugins. Sometimes these plugins are used to perform tasks that are usually done outside of webpack, such as cleaning the build directory or deploying the build. Webpack

8.10.1 How Webpack favor us?

1. Webpack gives you module system.
2. Thanks to webpack, we can import module from spesific place.
3. Webpack is bundling tool. It bundle CSS and Javascript and prepare well structured production environment.
4. With webpack we can use npm that solves dependency issues in ease.

55 56

[55] Latest version of webpack 5.0.0beta - Check: https://github.com/webpack/webpack/releases

[56] At its core, webpack is a static module bundler for modern JavaScript applications. When webpack rocesses your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.webpackbundle

8.11 LightHouse

8.11.1 What is lighthouse?

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO and more.

You can run Lighthouse in Chrome DevTools, from the command line, or as a Node module. You give Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining why the audit is important, as well as how to fix it.Lighthouse

8.11.2 What does lighthouse provide us?

First Contextual Paint: First Contentful Paint marks the time at which the first text or image is painted. contextual

First Meaningful Paint: First Meaningful Paint measures when the primary content of a page is visible.

Properly Size Image: Serve images that are appropriately-sized to save cellular data and improve load time. size

All text remains visible during webfont loads: Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. font

[user-scalable="no"] is not used in the ¡meta name="viewport">element and the [maximum-scale] attribute is not less than 5: Disabling zooming is problematic for users with low vision who rely on screen magnification to properly see the contents of a web page. zoom

Has a <meta name="viewport">tag with width or initial-scale: Add a '¡meta name="viewport">' tag to optimize your app for mobile screens.meta

Content is sized correctly for viewport: If the width of your app's content doesn't match the width of the viewport, your app might not be optimized for mobile screens.content

[57]

[57] **Lighthouse give more information about your page. Some restricted information are given here for being relevant to article.**

9 Web Page Rendering - Step by Step

- The DOM (Document Object Model) is formed from the HTML that is received from a server.
- Styles are loaded and parsed, forming the CSSOM (CSS Object Model).
- On top of DOM and CSSOM, a rendering tree is created, which is a set of objects to be rendered (Webkit calls each of those a "renderer" or "render object", while in Gecko it's a "frame"). Render tree reflects the DOM structure except for invisible elements (like the <head>tag or elements that have display:none; set). Each text string is represented in the rendering tree as a separate renderer. Each of the rendering objects contains its corresponding DOM object (or a text block) plus the calculated styles. In other words, the render tree describes the visual representation of a DOM.
- For each render tree element, its coordinates are calculated, which is called "layout". Browsers use a flow method which only required one pass to layout all the elements (tables require more than one pass).
- Finally, this gets actually displayed in a browser window, a process called "painting".

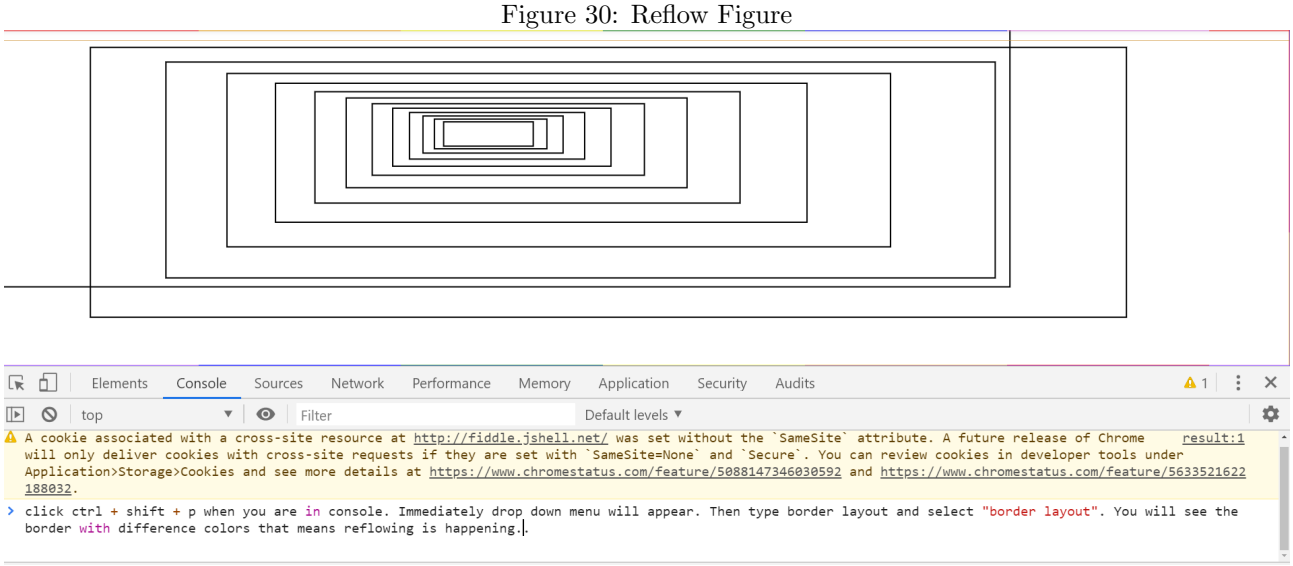
10 Experiment

In those examples, I aim to show how browser behaves in terms of reflow, reppaint and performance.

10.1 Reflow, Border Layout

Reflow is the name of the web browser process for re-calculating the positions and geometries of elements in the document, for the purpose of re-rendering part or all of the document. Because reflow is a user-blocking operation in the browser, it is useful for developers to understand how to improve reflow time and also to understand the effects of various document properties (DOM depth, CSS rule efficiency, different types of style changes) on reflow time. Sometimes reflowing a single element in the document may require reflowing its parent elements and also any elements which follow it.Why reflow is important

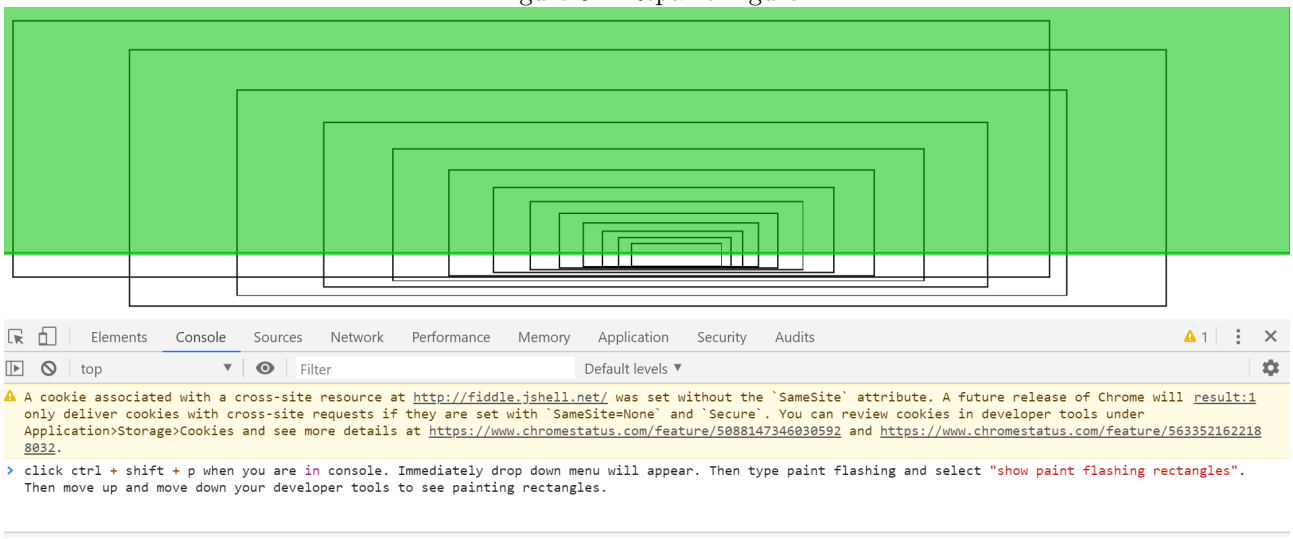
Figure 30: Reflow Figure



10.2 Repaint, Paint Flashing

Painting is the operation browser colorize elements pixel by pixel. Each reflow namely calculating layout may trigger re-paint.

Figure 31: Repaint Figure



58

[58] Example Code: https://jsfiddle.net/qzfj6xug/

10.3 Render Profiling

We will check the performangooce of the page. This test will give us necessary informations what brower handles in the page load. So we can concern experiment as a summary of that article. There is more than that actually. You can check [Lighthouse] section for that. I strongly recommend this.

Users expect pages to be interactive and smooth. Each stage in the pixel pipeline represents an opportunity to introduce jank. Learn about tools and strategies to identify and fix common problems that slow down runtime performance. Runtime Performance

Something there is you need to careful about. Firstly rule is "kiss", stick to that rule and you will be fine. Otherwise you will find yourself feel hadhache. I will enlist what you shouldn't do.

1. Dont force javascript the browser to calculate steps that is discussed above.
2. Dont overenginner about your css. Follow block element modifier principle.
3. Watch out painting bottlenecks

Lets do profilling.

10.3.1 Panes

Figure 32: Flow Chart Diagram

- First pane is for FPS (frame per second)
- Second pane is for CPU usage
- Third one is for layout - painting and composite

Figure 33: Flow Chart Diagram

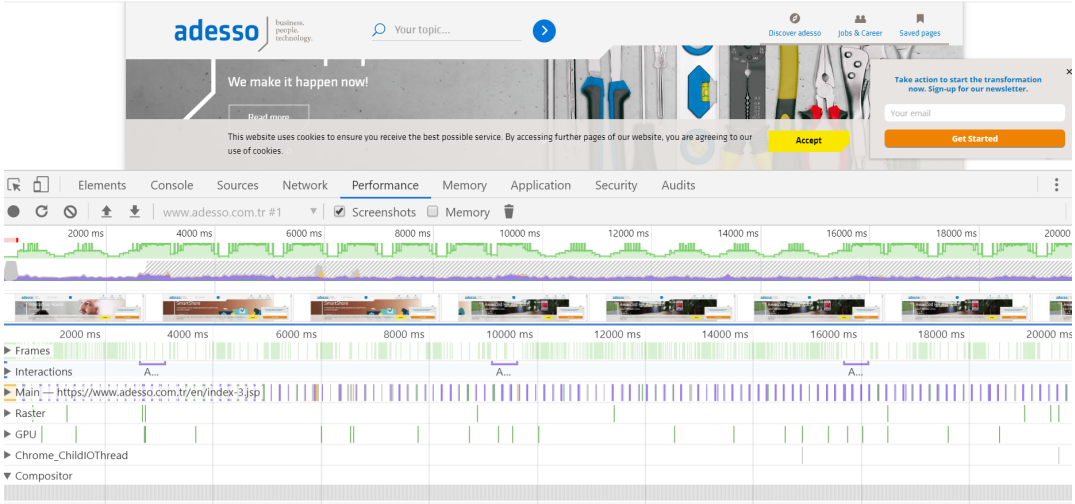
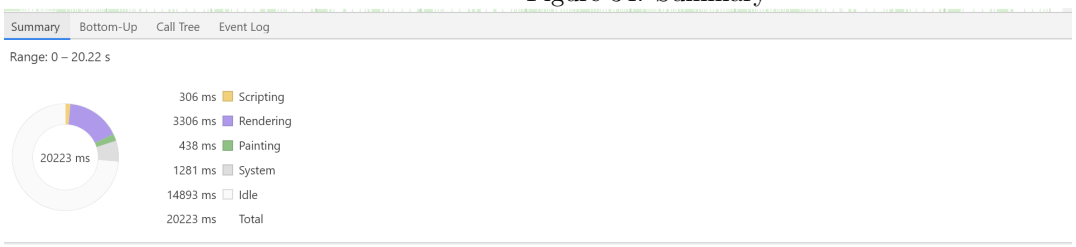


Figure 34: Summary



59 60

[59] Check: https://httparchive.org/reports/page-weight

[60] Related Reading: https://developers.google.com/web/fundamentals/performance/rendering/reduce-the-scope-and-complexity-of-style-calculations

11 Conclusion

This article can be evaluated as a semantic roadmap about rendering page. It is possible some issues are missed here. It is not really plausible to guarantee full coverage about how browser render the page to be honest. It is really broad topic. The motivation here, make developer step into this ecosystem.

It is suggested to you to check "Suggested sources and reading sections". In there you can find lots of information about the importantce that I try to emphasize here.

It is possible to find some real world examples. Thanks to that, It is easy to understand gist of what we try to handle here. Not last but least, The advanced content give some tips about how we combine techniques and terminologies in real world.

12 Appendix

12.1 Suggested Sources & Readings

Web Stadard

- https://www.w3.org/

Web Community

- https://whatwg.org/

Web Docs

- https://developer.mozilla.org/en-US/docs/Web
- https://webplatform.github.io/
- https://css-tricks.com/
- http://html5doctor.com/
- https://html5please.com/
- https://developers.google.com/web
- https://www.flexboxpatterns.com/
- https://maintainablecss.com/
- http://learnlayout.com/
- https://www.w3.org/wiki/Traversing_the_DOM#Growing_trees
- https://www.youtube.com/user/DevTipsForDesigners/playlists?shelf_id=0&view=1&sort=dd
- https://www.youtube.com/watch?v=0xIDLw0M-m0&list=PL4cUxeGkcC9ij8CfkAY2RAGb-tmkNwQHG
- https://itnext.io/how-the-browser-renders-a-web-page-dom-cssom-and-rendering-df10531c9969

Check Support

- https://caniuse.com/

Validator

- https://validator.w3.org/
- https://jigsaw.w3.org/css-validator/

Inspector

- http://articles.asmcbain.net/articles/inspector/
- https://debugbrowser.com/
- https://developers.google.com/web/tools/chrome-devtools?utm_source=dcc&utm_medium=redirect&utm_campaign=2018Q2
- https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/samples/bg182326(v=vs.85)?redirectedfrom=MSDN
- https://support.apple.com/tr-tr/guide/safari-developer/welcome/mac
- https://developer.mozilla.org/en-US/docs/Tools

Betterment

- http://jslint.com/
- http://csslint.net/

- https://beautifier.io/
- https://github.com/chriswrightdesign/websites-for-non-jerks
- https://gist.github.com/paulirish/5d52fb081b3570c81e3a

Color

- https://snook.ca/technical/colour_contrast/colour.html#fg=33FF33,bg=333333
- https://contrast-ratio.com/

Accessibility

- https://a11yproject.com/

Coding Style

- https://github.com/airbnb

User Style

- https://github.com/amcgregor/userstyles#readme

Problem Solving

- https://stackoverflow.com/help/minimal-reproducible-example
- http://xyproblem.info/

Tools & Projects

- https://www.quicklycode.com/?s=css
- https://plainjs.com/
- https://gohugo.io/

12.2 Inspired From

1. https://webplatform.github.io/docs/guides/doctypes_and_markup_styles/
2. https://www.w3.org/wiki/Doctypes_and_markup_styles
3. https://www.valuecoders.com/blog/technology-and-apps/top-10-advantages-of-html5-development/
4. https://blog.jakoblind.no/whats-the-advantage-with-webpack/
5. https://scotch.io/bar-talk/native-lazy-loading-launched-on-chrome-76
6. https://hub.packtpub.com/google-chrome-76-now-supports-native-lazy-loading/
7. https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API
8. https://frontarm.com/james-k-nelson/static-vs-server-rendering/
9. https://www.smashingmagazine.com/2016/02/preload-what-is-it-good-for/

13 Referrences

1. https://www.geeksforgeeks.org/html-viewport-meta-tag-for-responsive-web-design/
2. https://webpack.js.org/
3. https://developer.mozilla.org/en-US/docs/Web/HTML/Quirks_Mode_and_Standards_Mode
4. https://developers.google.com/web/tools/chrome-devtools/open
5. https://tinyurl.com/u5jdwyq
6. https://html.spec.whatwg.org/multipage/semantics.html#semantics
7. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
8. https://developer.mozilla.org/en-US/docs/Web/HTML/Quirks_Mode_and_Standards_Mode
9. http://f.cl.ly/items/0P0g0B3U2y2H0y3E1E38/Screen+Shot+2017-07-30+at+05.15.37.png
10. https://developers.google.com/web/tools/lighthouse/
11. https://web.dev/first-contentful-paint/?utm_source=lighthouse&utm_medium=devtools
12. https://web.dev/first-meaningful-paint/?utm_source=lighthouse&utm_medium=devtools
13. https://web.dev/font-display/?utm_source=lighthouse&utm_medium=devtools
14. https://web.dev/meta-viewport/?utm_source=lighthouse&utm_medium=devtools
15. https://web.dev/uses-responsive-images/?utm_source=lighthouse&utm_medium=devtools
16. https://web.dev/viewport/?utm_source=lighthouse&utm_medium=devtools
17. https://web.dev/content-width/?utm_source=lighthouse&utm_medium=devtools
18. https://webpack.js.org/concepts/
19. https://codepen.io/eitanp461/pen/BRLZYq/
20. https://web.dev/native-lazy-loading/
21. https://scotch.io/bar-talk/native-lazy-loading-launched-on-chrome-76
22. https://developers.google.com/web/tools/chrome-devtools/rendering-tools
23. https://tinyurl.com/qtzr5h4
24. https://tinyurl.com/wgtn3z5
25. https://css-tricks.com/almanac/properties/w/will-change/#:~:text=The%20will%2Dchange%20property%20in,browser%20optimizations%20will%20be%20applied.
26. https://css-tricks.com/why-would-i-use-a-webpack/
27. https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5
28. https://hacks.mozilla.org/2017/09/building-the-dom-faster-speculative-parsing-async-defer-and-pre
29. https://www.smashingmagazine.com/2016/02/preload-what-is-it-good-for/
30. https://w3c.github.io/resource-hints/#prefetch
31. https://html.spec.whatwg.org/multipage/workers.html#workers
32. https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely
33. https://developers.google.com/web/fundamentals/performance/rendering/avoid-large-complex-layouts-a
34. https://mattstauffer.com/blog/organizing-css-oocss-smacss-and-bem/
35. about-normalize-css

36. https://web.dev/extract-critical-css/

37. https://en.wikipedia.org/wiki/Above_the_fold

38. google-chrome-76-now-supports-native-lazy-loading

39. https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API